Project 4 Questions

Question 1:

Compare kern/mpentry.S side by side with boot/boot.S.
Bearing in mind that kern/mpentry.S is compiled and linked to run above KERNBASE just like everything else in the kernel,
what is the purpose of macro MPBOOTPHYS?
Why is it necessary in kern/mpentry.S but not in boot/boot.S?
In other words, what could go wrong if it were omitted in kern/mpentry.S?

Hint: recall the differences between the link address and the load address that we have discussed in Project 1.

- Both "mpentry.S" and "boot.S" are similar, except that "boot.S" uses A20 which is meant for backwards compatibility on early PCs that wrap around addresses higher than 1 MB.
- Instead "mpentry.S" uses MBOOTPHYS to calculate the physical address from the virtual address and producing the offset.
- Without MBOOTPHYS, mpentry would fail to load the correct address because it is not depending on a linker on the KERNBASE like "boot.S".

Question 2:

It seems that using the big kernel lock guarantees that only one CPU can run the kernel code at a time.
Why do we still need separate kernel stacks for each CPU?
Describe a scenario in which using a shared kernel stack will go wrong, even with the protection of the big kernel lock.

- The kernel lock helps protect against a race condition. If CPU 1 entered the kernel, but then CPU 2 also enters, once CPU 1 has completed it's task it will knock out the trap frame for CPU 2. That will then prevent CPU 2 from leaving the kernel because it can not clear out it's own trap frame and then CPU 1 is still locked since it will believe it is still using the kernel since the trap frame for CPU 1 is present.

Quesstion 3:

In your implementation of env_run() you should have called lcr3().

Before and after the call to lcr3(), your code makes references (at least it should) to the variable e, the argument to env_run.
Upon loading the %cr3 register, the addressing context used by the MMU is instantly changed. But a virtual address (namely e) has meaning relative to a given address context--the address context specifies the physical address to which the virtual address maps.
Why can the pointer e be dereferenced both before and after the addressing switch?

- The memory above UTOP is mapped to the same physical address in each enviroment. This allows the kernel to make syscalls using the mapped memory.

Question 4:

Whenever the kernel switches from one environment to another, it must ensure the old environment's registers are saved so they can be restored properly later. Why? Where does this happen?

- If the registers fail to properly save the old enviroment, the system will crash because it can not revert after the interrupt is completed. The registers should be saved during the switch from user mode to kernel mode in the trap frame.
- This is seen in "trap.c" within the trap function when "curenv->env_tf = *tf" runs to copy the trap frame, so that the running enviroment will restart at the trap point.